Australian Government
Department of Defence

Defence Signals Directorate

Australasian Information Security Evaluation Program

Minimisation of complexity (ADV_INT.3) - CC V2.2

Common Evaluation Methodology

15 February 2006

Version 1.2

Amendment Record

Version	Date	Description
1.0	12 September 2005	Release
1.1	18 November 2005	Releasable to AISEFs
1.2	15 February 2006	Added CC V2.2 to title

Table of Contents

1 Minimisation of complexity (ADV_INT.3)

1.1 Objectives

1 This objective of this sub-activity is to determine whether the developer has implemented a TSF that minimises complexity. While the primary focus of this family is to require the developer to design and implement a TSF that is well understood by the development staff, this component also requires the design and implementation to be easily understood by an independent third-party in an efficient manner. This translates into requirements such as the justification for any unused or redundant implementation, and the elimination of non TSP-enforcing modules.

1.2 Application notes

2 The role of the architectural description is to provide evidence of modularity of the TSF and to justify the complexity of the design and implementation. The focus of the architectural description is on how the TSF has been designed and implemented using sound software and/or hardware engineering principles leading to a well-designed and secure TSF. The architectural description provides information regarding the appropriateness of functions included in each module of the low-level design, and on the interaction between the modules. The low-level design (ADV_LLD), on the other hand, describes the functionality of the modules of the TSF and how those modules work to satisfy the SFRs.

3 The modules identified in the architectural description are the same as the modules identified in the low-level design. Some of the information from the low-level design may be applicable in meeting the required content for the architectural description, and can be included by reference.

4 For portions of the TSF implemented in software or firmware, a module consists of one or more source code files that cannot be decomposed into smaller compilable units. For portions of the TSF implemented in hardware, a module might consist of a grouping of components providing TSP-enforcement functionality.

1.3 Input

5 The evaluation evidence for this sub-activity is:

a) the ST;

b) the low-level design;

c) the implementation representation;

d) the architectural description.

1.4 Evaluator actions

6 This sub-activity comprises three CC Part 3 evaluator action elements:

a) ADV_INT.3.1E;

b) ADV_INT.3.2E;

c) ADV_INT.3.3E

1.4.1 ADV_INT.3.1E

ADV_INT.3.1C	The architectural description shall identify the modules of the TSF.

ADV_INT.3-1 The evaluator ***shall check*** the architectural description to determine that the identification of modules is identical to the modules that are described in the low-level design.

7 The evaluator determines that the modules identified and described in the architectural description are the exact same modules identified in the low-level design. Any modules that are identified in either document that are not present in both signify a failure of this work unit.

ADV_INT.3.2C	The architectural description shall describe the purpose, interface, parameters, and effects of each module of the TSF.

ADV_INT.3-2 The evaluator ***shall check*** the architectural description to determine that it describes the type of coupling between modules of the TSF in terms of purpose, parameters and effects.

8 For the purposes of this sub-activity, coupling is defined as the manner and degree of interdependence between software modules; types of coupling include call, common and content coupling. These types of coupling are characterised below, listed in the order of decreasing desirability:

a) **call**: two modules are call coupled if they communicate strictly through the use of their documented function calls; examples of call coupling are data, stamp, and control, which are defined below.

i) **data**: two modules are data coupled if they communicate strictly through the use of call parameters that represent single data items.

ii) **stamp**: two modules are stamp coupled if they communicate through the use of call parameters that comprise multiple fields or that have meaningful internal structures.

iii) **control**: two modules are control coupled if one passes information that is intended to influence the internal logic of the other.

b) **common**: two modules are common coupled if they share a common data area or a common system resource. Global variables indicate that modules using those global variables are common coupled. Common coupling through global variables is generally allowed, but only to a limited degree. For example, variables that are placed into a global area, but are used by only a single module, are inappropriately placed, and should be removed. Other factors that need to be considered in assessing the suitability of global variables are:

i) The number of modules that modify a global variable: In general, only a single module should be allocated the responsibility for controlling the contents of a global variable, but there may be situations in which a second module may share that responsibility; in such a case, sufficient justification must be provided. It is unacceptable for this responsibility to be shared by more than two modules. (In making this assessment, care should be given to determining the module actually responsible for the contents of the variable; for example, if a single routine is used to modify the variable, but that routine simply performs the modification requested by its caller, it is the calling module that is responsible, and there may be more than one such module). Further, as part of the complexity determination, if two modules are responsible for the contents of a global variable, there should be clear indications of how the modifications are coordinated between them.

ii) The number of modules that reference a global variable: Although there is generally no limit on the number of modules that reference a global variable, cases in which many modules make such a reference should be examined for validity and necessity.

c) **content**: two modules are content coupled if one can make direct reference to the internals of the other (e.g. modifying code of, or referencing labels internal to, the other module). The result is that some or all of the content of one module are effectively included in the other. Content coupling can be thought of as using unadvertised module interfaces; this is in contrast to call coupling, which uses only advertised module interfaces.

9 The evaluator looks for an analysis of the coupling (interaction) between modules of the TSF. The evaluator ensures that the analysis correctly identifies and describes all module interactions by comparing it with the evaluators understanding of interactions based on the implementation representation. The coupling analysis includes any explicit calls made by functions in a module to other modules, the passing of pointers from one function to another function, the use of global variables, shared memory, message passing, or any other means that functions may use to communicate with other modules.

10 The evaluator also uses the implementation representation to ensure modules do not reference global variables that are not contained in the coupling analysis and that the references modules make (e.g., read, write) are consistent with the references specified in the coupling analysis.

11 If the TSF contains no software or firmware modules then this work unit is not applicable and considered to be satisfied.

ADV_INT.3-3 The evaluator ***shall check*** the architectural description to determine that it describes the type of cohesion between modules of the TSF in terms of purpose, parameters and effects.

12 For this sub-activity, cohesion is defined as the manner and degree to which the tasks performed by a module are related to one another; types of cohesion include coincidental, communicational, functional, logical, sequential, and temporal. These types of cohesion are characterised below, listed in the order of decreasing desirability and are equally applicable to software, firmware and hardware modules of the TSF.

a) **functional cohesion**. A module with this characteristic performs activities related to a single purpose. A functionally cohesive module transforms a single type of input into a single type of output, such as a stack manager or a queue manager.

b) **sequential cohesion**. A module with this characteristic contains functions each of whose output is input for the following function in the module. An example of a sequentially cohesive module is one that contains the functions to write audit records and to maintain a running count of the accumulated number of audit violations of a specified type.

c) **communicational cohesion**. A module with this characteristic contains functions that produce output for, or use output from, other functions within the module. An example of a communicationally cohesive module is an access check module that includes mandatory, discretionary, and capability checks.

d) **temporal cohesion**. A module with this characteristic contains functions that need to be executed at about the same time. Examples of temporally cohesive modules include initialisation, recovery, and shutdown modules.

e) **logical (or procedural) cohesion**. A module with this characteristic performs similar activities on different data structures. A module exhibits logical cohesion if its functions perform related, but different, operations on different inputs.

f) **coincidental cohesion**. A module with this characteristic performs unrelated, or loosely related, activities.

13 The evaluator looks for an analysis of cohesion between modules of the TSF and ensures that it is complete in that each module is associated with the type of cohesion that is exhibited, as well as a description of how the determination of the type of cohesion was made by the developer.

ADV_INT.3.3C The architectural description shall describe how the TSF design provides for largely independent modules that avoid unnecessary interactions.

ADV_INT.3-4 The evaluator ***shall examine*** the architectural description to determine that the developer's process for modular decomposition results in a modular TSF low-level design.

14 Modules interact by providing services to one another, or by cooperating in achieving an overall goal.

15 The evaluator assesses the developer's design methodology to determine that the TSF design incorporates sound engineering principles to decompose the TSF into modules. Modules should be designed to include only related functions, to have a single, well-defined purpose, and to avoid unnecessary interactions with other modules. The methodology should describe what are acceptable forms of communication between modules and under what circumstances discouraged forms of coupling and cohesion are considered acceptable. The methodology should also address complexity and what steps are taken to ensure modules are not overly complex. The intent here is that the developer's process should cover/address the developer action elements for the TSF internals (ADV_INT) requirements and this process provides guidance to the developer on how to design and implement their modules.

16 The methodology provided by the developer provides the evaluator with an understanding of the developer's approach to designing (or restructuring of) the TSF and aids the evaluator in performing other work units for this component.

ADV_INT.3.4C The architectural description shall describe the layering architecture.

ADV_INT.3-5 The evaluator ***shall examine*** the architectural description to determine that the layering architecture is described.

17 The evaluator ensures the architectural description provides a presentation that illustrates the layers within the TSF and describes the purpose of each of the layers. Typically a figure or diagram will show the layers and their hierarchy, and at a high level, interactions between the layers. The purpose of each layer provides enough detail that makes clear each layer's role in the architecture, and allows the evaluator to determine how the modules associated with each layer work together to meet that purpose.

ADV_INT.3-6 The evaluator ***shall examine*** the architectural description to determine that the services provided with each layer is consistent with the description of the layering architecture.

18 The evaluator ensures that the description of the services provided by each layer is consistent with the descriptions of the layers. The description of each layer's purpose and role maps to the services that each layer is providing. An inconsistent description would exist if a service is being exported to another layer, but there is no indication in the purpose of the layer exporting the service that would suggest the layer is providing such a service.

ADV_INT.3-7 The evaluator ***shall examine*** the architectural description to determine that it provides a description of the developer's methodology in determining the TSF's layered architecture.

19 The description of the layering architecture provides the evaluator an understanding of how the layered architecture was determined. It describes the process or methodology the developer used in determining the layered architecture, and provides confidence that the layered architecture resulted from sound security engineering principles, rather than an ad-hoc grouping of modules into layers. The methodology should describe how the requirements associated with layering in this component were considered, and any design decisions that were considered to satisfy the requirements or required exceptions to the principles of a sound layered architecture.

ADV_INT.3-8 The evaluator ***shall check*** the architectural description to ensure it identifies all of the modules associated with each layer.

20 The evaluator ensures that all modules a in the architectural description are associated with a layer and that a module is not associated with more than one layer. The evaluator will want to perform the work unit determining that there is a one-to-one correspondence with the modules identified in the architectural description and the low-level design description before performing this work unit to ensure all modules are correctly identified, as well as associated with a layer. There should be no instances of a module not being associated with a layer or a module associated with multiple layers.

ADV_INT.3-9 The evaluator ***shall examine*** the architectural description and the low-level design to determine that the modules associated with each layer is consistent with each layer's role.

21 The evaluator ensures that the modules associated with each layer is appropriate and consistent with the activities the layer performs and the services it provides. By consulting the functional description of modules presented in the modular design, and reviewing the purpose/role of a layer, the evaluator determines that the modules associated with a layer are appropriate and work together to achieve the functionality presented by a layer.

22 If a module is associated with a layer and it is not clear from the modules description presented in the modular design description how the module supports or satisfies the purpose of a layer, there may be an inadequacy in the layer's description, there may be a concern regarding other requirements (e.g., interactions between layers), or the module may be simply misplaced and some form of corrective action is necessary.

ADV_INT.3-10 The evaluator ***shall check*** the architectural description to determine the interactions between layers of the TSF are identified.

23 The evaluator ensures that the interactions between each layer have been identified in the architectural description. It is not necessary for the architectural description to repeat the detailed information of the interaction, a reference to where the details of the interaction (whether it be in the low-level design description, architectural description as part of the developer's coupling analysis. or the functional specification) is sufficient.

ADV_INT.3-11 The evaluator ***shall examine*** the architectural description to determine that the interactions between layers are accurately and completely described.

24 The evaluator uses coupling information contained in the architectural description to determine the interactions between layers is consistently (accurately and completely) described. The evaluator considers calls that are made from modules that reside in a layer to functions/routines contained in modules that reside in another layer. The evaluator also considers other means in which modules communicate (e.g., shared memory, global variables, passing of function pointers). All forms of module communication are presented in the developer's coupling analysis, and for this work unit the evaluator is only concerned with the module interactions that take place between layers (i.e., the evaluator need not be concerned with interactions between modules that reside in the same layer).

25 The evaluator ensures that the description of layer interactions make it clear which layer is initiating the call.

ADV_INT.3.5C The architectural description shall show that mutual interactions have been minimised, and justify those that remain.

ADV_INT.3-12 The evaluator ***shall examine*** the architectural description to determine if the justification for coupling and cohesion between modules other than the ones permitted is reasonable.

26 The evaluator judges the developer's rationale for modules that exhibit coupling and cohesion other than those allowed and determines if the justifications are reasonable. The justification is provided for each module that is not conformant to the allowed forms of coupling and cohesion.

27 In order to assess the justification for either coupling or cohesion forms other than those permitted may require the evaluator to examine the implementation representation. The evaluator confirm the justification is reasonable, by understanding the offending module, and determining that requiring the module to strictly conform to the coupling or cohesion standards would be unreasonable. Through this work unit, the evaluator gains an understanding of how and why the developer has minimised unnecessary interactions.

ADV_INT.3-13 The evaluator ***shall check*** the architectural description to determine it contains a justification of interactions between layers that are initiated from a lower layer to a higher layer.

28 Since interactions that are initiated from a lower layer to a higher layer indicate an undesirable dependency, the evaluator ensures that for each interaction that is initiated from a layer to a layer higher in the hierarchy a justification exists. The evaluator uses the information garnered from the previous work unit to determine the set of initiated interactions of interest.

ADV_INT.3-14 The evaluator ***shall examine*** the architectural description to determine the justifications for lower layers initiating an interaction to a higher layer are acceptable.

29 The are instances where it may be acceptable for layering violations (for a lower layer to have a dependency on a higher layer). Typically, these exceptions are for efficiency or performance reasons. The evaluator assesses the developer's justification and exercises judgement to the acceptability of the argument. The evaluator considers how pervasive are the layering violations, does the justification for the laying violation make sense in the context of the developer's architecture, or does it appear that modest alterations could be made to the architecture resulting in the elimination of the violation without impacting performance too much. The evaluator considers the trade-off between understandable (and maintainable) design and the need for efficiency and performance.

30 If there are no interactions from lower to higher layers, then this work unit is not applicable and is considered to be satisfied.

ADV_INT.3.6C The architectural description shall describe how the entire TSF has been structured to minimise complexity.

ADV_INT.3-15 The evaluator ***shall examine*** the architectural description to determine the developer's description for how the TSF is designed and structured minimises complexity is acceptable.

31 The evaluator uses the information provided in the architectural description to make an assessment of the developer's claim to have structured the TSF to minimise complexity. Here the developer provides a description that ties all of the developer's activities and analyses together to provide a "big picture" view of how the TSF has been designed and structured. This argument should provide specific references to the detailed analyses performed, the methodologies employed, and provide the "glue" that "tells a story" of how all of the performed activities resulted in a well structured TSF that has minimised complexity and is simple enough to be analysed.

ADV_INT.3.7C The architectural description shall justify the inclusion of any non TSP-enforcing modules in the TSF.

ADV_INT.3-16 The evaluator ***shall examine*** the architectural description to determine the justifications for including non TSP-enforcing modules are acceptable.

32 The evaluator considers the justifications made by the developer as to why non-TSP enforcing modules are included in the TSF. At this level of assurance, it is difficult to provide examples of where non TSP-enforcing modules that this not SFR-supporting or SFR-enforcing should be allowed in the TSF. The evaluator uses judgement in determining the appropriateness of the developer's argument while taking into account that the TSF's complexity is to be minimal at this level of assurance.

1.4.2 ADV_INT.3.2E

ADV_INT.3.2E	The evaluator shall determine that both the low-level design and the implementation representation are in compliance with the architectural description.

ADV_INT.3-17 The evaluator ***shall examine*** the architectural description to determine that the TSF's low-level design is an accurate reflection of the developer's decomposition process.

33 The evaluator compares the developer's documented process for decomposition presented in the architectural description with the low-level design to determine if the developer followed their process. The evaluator will need to perform this work unit in conjunction with other work units. In fact, the evaluator should understand the process the developer has employed for decomposing their TSF into modules and while performing other work units ensure that process was followed.

ADV_INT.3-18 The evaluator ***shall examine*** the architectural description to determine that the implementation representation is in compliance with the architectural description.

34 The evaluator looks for consistency between the analysis of modularity (coupling and cohesion) in the architectural description and the implementation representation (source code and/or hardware drawings) of the TSF. Correspondence information between low-level design and implementation may support this consistency check, but this consistency check is focused on the aspects, that the modularity concept outlined in the architectural description is not made ineffective by the way the implementation level is structured.

ADV_INT.3-19 The evaluator ***shall examine*** the implementation representation to determine that the developer has adhered to the coding standards for the TSF modules.

35 The evaluator ensures that the development of the implementation representation adhered to the coding standards that are included in the development tools (Tools and techniques (ALC_TAT)). A developer may have more than one coding standard for different components of the TOE, and this is acceptable, as long as each standard contains the necessary information. This is accomplished by determining that the format prescribed by the coding standards (e.g., comments, naming conventions, indentation) is exhibited in the implementation representation. The evaluator ensures that the guidelines for programming logic (e.g., types of loops, control flow, error handling, the use of pointers) are exhibited in the implementation representation. The evaluator determines that other aspects of code development that are defined in the coding standards are followed by the developer.

ADV_INT.3-20 The evaluator ***shall examine*** the architectural description to determine if the justification for deviations from the coding standards for the implementation representation is reasonable.

36 There may be acceptable occurrences where the implementation representation does not follow the coding standards. A justification that is solely based on the use of third-party source code or source code that was inherited from previous life-cycles of the TOE is not acceptable. Typically, deviations from the prescribed formatting aspects of the coding standards are not justifiable. Acceptable justification for deviation from the coding standards would generally be related to performance or efficiency reasons, and possibly for backward compatibility.

ADV_INT.3-21 The evaluator ***shall examine*** the implementation representation to determine that the type of coupling exhibited by the TSF modules is consistent with that claimed in the architectural description.

37 The evaluator makes an independent assessment of the type of coupling that exists between modules and ensures that they arrive at the same conclusion as the presented in the developer's analysis.

ADV_INT.3-22 The evaluator ***shall examine*** the implementation representation to determine that the type of cohesion exhibited by the TSF modules is consistent with that claimed in the architectural description.

38 Cohesion is a measure of the strength of relationship between the activities performed by a module. Cohesion ranges from the most desirable form, functional cohesion, to the least desirable form, coincidental cohesion. Allowable forms of cohesion include functional, sequential, and communicational cohesion; coincidental cohesion is unacceptable. Temporal cohesion is acceptable only for special-purpose use, such as initialisation, shutdown, and error recovery. Logical cohesion may be acceptable, based on the argument for its justification.

39 The evaluator uses the implementation representation and low-level design to determine the purpose of the modules and the relationship the functions have within a module to achieve that purpose. The processing performed by the functions in a module and the service they provide are an indication of the design in the grouping of functions and what type of cohesion is exhibited.

40 The evaluator performs an analysis to determine the type of cohesion exhibited by the modules and ensures that the evaluator's findings are consistent with the developer's claim in the architectural description. It is important to note that there may be some subjectivity involved in making a determination of the type of cohesion exhibited (e.g., sequential vs. temporal), but the type of cohesion that is deemed unacceptable (e.g., coincidental, logical) is more clear cut.

41 As with all areas where the evaluator performs activities on a subset the evaluator provides a justification of their sample size and scope.

1.4.3 ADV_INT.3.3E

ADV_INT.3.3E	**The evaluator shall confirm that the portions of the TSF that enforce any access control and/or information flow control policies are simple enough to be analysed.**

ADV_INT.3-23 The evaluator ***shall examine*** the architectural description and implementation representation to determine the modules of the TSF that enforce access control and/or information flow control policies are simple enough for analysis.

42 A metric that simply relies on the numbers of lines of code or the number of components on a hardware drawing, or the percentage of comments and statements is considered a weak complexity measure. For software more meaningful metrics consider the number of unique operators and operands in the program, or which measures the amount of unstructured decision logic in software by examining cyclomatic complexity. For a hardware TOE, consider the types of components to be used for implementation – use of simple logic gates over more complex multi functional and programmable logic arrays.

43 The evaluator determines the complexity by measuring the length of time it takes the evaluator to understand the logic in a module and the module's relationship with other modules. When using the design documentation and architectural description, the evaluator should understand a module's implementation and interactions in a reasonable amount of time. The measurement of time is subjective and is based on the evaluator's experience and skill in reading and understanding the type of TOE. It is assumed that the evaluator has knowledge of computer programming, or hardware engineering, and is familiar with the implementation standards and languages used by the developer. Given this minimum level of expertise, the amount of time it takes to understand a module's implementation should not exceed eight hours.

44 The evaluator needs to also consider how a module incorporates access to "global entities", such as functions, macros or data. One example of this is the use of header files in the C programming language. Often times header files are nested and a programmer does not understand all of the contents that are potentially accessible by their software. Another poor example of programming is the incorporation of header files without knowing whether the header file is actually needed. The problem arises when a programmer makes a mistake (e.g., mis-types a variable or macro name). The likelihood of the improper variable being referenced or macro being invoked is less if only the necessary header files are included. Having several or more unnecessary header files included increases the likelihood that the mis-typed variable or macro exists in one of those unnecessary header files which could lead to serious consequences.